technologies

MDPI

Article

A Comparative Study of Machine and Deep Learning Approaches for Smart Contract Vulnerability Detection

Mohammed Yaseen Alhayani [1,*], Wisam Hazim Gwad [2], Shahab Wahhab Kareem [3,4] and Moustafa Fayad [5,*]

[1] Department of Software, Information Technology College, Ninevah University, Mosul 41001, Iraq
[2] Department of Artificial Intelligence Technology, College of Technical Engineering, Alnoor University, Mosul 41012, Iraq; wisam.hazim@alnoor.edu.iq
[3] Department of Artificial Intelligence and Robotics Engineering, Technical College of Computer and Informatics Engineering, Erbil Polytechnic University, Erbil 44001, Iraq
[4] Department of Information Technology, College of Engineering and Computer Science, Lebanese French University, Erbil 44001, Iraq
[5] Université Marie et Louis Pasteur, SINERGIES (UR 4662), 25030 Besançon, France
* Correspondence: mohammed.y.abdullah@uoninevah.edu.iq (M.Y.A.); moustafa.fayad@univ-fcomte.fr (M.F.)

Abstract

The increasing use of blockchain smart contracts has introduced new security challenges, as small coding errors can lead to major financial losses. While rule-based static analyzers remain the most common detection tools, their limited adaptability often results in false positives and outdated vulnerability patterns. This study presents a comprehensive comparative analysis of machine learning (ML) and deep learning (DL) methods for smart contract vulnerability detection using the BCCC-SCsVuls-2024 benchmark dataset. Six models (Random Forest, k-Nearest Neighbors, Simple and Deep Multilayer Perceptron, and Simple and Deep one-dimensional Convolutional Neural Networks) were evaluated under a unified experimental framework combining RobustScaler normalization and Principal Component Analysis (PCA) for dimensionality reduction. Our experimental results from a five-fold cross-validation show that the Random Forest classifier achieved the best overall performance with an accuracy of 89.44% and an F1-score of 93.20%, outperforming both traditional and neural models in stability and generalization. PCA-based feature analysis revealed that opcode-level features, particularly stack and memory manipulation instructions (PUSH, DUP, SWAP, and RETURNDATASIZE), were the most influential in defining contract behavior.

Keywords: smart contracts; vulnerability detection; machine learning; deep learning

Academic Editor: Martin Valtierra-Rodriguez

Received: 16 November 2025
Revised: 14 December 2025
Accepted: 15 December 2025
Published: 16 December 2025

Citation: Alhayani, M.Y.; Gwad, W.H.; Kareem, S.W.; Fayad, M. A Comparative Study of Machine and Deep Learning Approaches for Smart Contract Vulnerability Detection. *Technologies* **2025**, *13*, 592. https://doi.org/10.3390/technologies13120592

1. Introduction

Since the beginning, economic exchanges between entities have been governed by traditional contracts, which formalize the terms of the agreement between the parties. Modern scientific and technological advances have transformed this conventional mechanism. Blockchain is one of the most significant recent innovations: it acts as a distributed register that ensures trust, transparency, and immutability of data [1]. With the emergence of Ethereum, blockchain technology gave rise to smart contracts [2]. The concept of smart contracts was introduced by Nick Szabo back in the 1990s as a digital alternative to traditional paper-based agreements [3].

Smart contracts are software applications deployed on a blockchain [4]. As such, they may contain critical vulnerabilities. Several cyberattacks have exposed these flaws, leading

to significant and sometimes unpredictable financial losses. A well-known example is the DAO hack in 2016, which exploited reentrancy vulnerability, allowing attackers to steal 3.6 million Ether, worth around 60 million dollars [5]. Indeed, even a minor bug in a smart contract handling substantial funds can result in significant losses. Therefore, the early detection and correction of vulnerabilities have become a necessity.

The identification of security vulnerabilities still largely relies on traditional methods such as manual inspection, code auditing, and even formal verification. Tools like Oyente, Mythril, Securify, Slither, SmartCheck, etc., automate the static analysis of contracts and target common types of vulnerabilities (reentrancy, time dependency, incorrect authorization with tx.origin, unhandled exceptions, etc.) [6]. However, these rule-based tools often generate false positives or false negatives since they cannot fully capture the complex logic of contracts. Moreover, static rules quickly become outdated in the face of rapidly evolving code patterns [5,7].

In this context, artificial intelligence emerges as a promising solution to enhance the security of smart contracts. AI-based approaches learn directly from data and can continuously adapt to new types of vulnerability.

The main contributions of this paper are:

- We present a unified and comprehensive evaluation of classical machine learning models (Random Forest (RF) and k-Nearest Neighbors (KNN)) and deep learning architectures (Multilayer Perceptron (MLP) and one-dimensional Convolutional Neural Networks (CNN1D)) for smart-contract vulnerability detection using the large-scale BCCC-SCsVuls-2024 benchmark.
- We investigate the features that most influence the classification process by analyzing the contribution of original feature groups to the first principal components obtained through PCA.

The remaining part of this manuscript is organized as follows. Section 2 reviews the principal related works on smart-contract security, including static analysis tools, machine-learning techniques, and deep-learning approaches. Section 3 introduces the BCCC-SCsVuls-2024 dataset and describes its vulnerability classes and feature categories. Section 4 presents the data-preprocessing pipeline and the experimental framework, detailing the scaling procedure, PCA-based dimensionality reduction, and the configuration of the six evaluated models. Section 5 reports and discusses the experimental results, comparing model performance and examining PCA-derived feature contributions. Finally, Section 6 concludes the study and outlines future research directions, including multi-label vulnerability detection and real-time analysis frameworks.

2. Related Work

In this section we present the main contributions of automated vulnerability detection, focusing on both classical program analysis tools and machine/deep learning-based approaches, and describe the research gaps that motivate our study.

Static and Automated Analysis Tools Early research primarily counted on automated static analysis tools to detect vulnerabilities in Solidity smart contracts. Durieux et al. [8] conducted one of the largest empirical evaluations of such tools, applying nine widely used analyzers to two newly curated datasets through the SmartBugs framework. They found the limited reliability of static analysis, with only 42% of vulnerabilities always being detected (in addition to tools reporting many false positives). Ghaleb et al. [9] also developed a fault injection framework named SolidiFI to evaluate static analyzers. By systematically injecting security bugs into contracts, they demonstrated that most tools fail to detect bugs they claimed to cover. These studies underscored both the utility and limitations of static analyzers: while valuable for initial detection, their high false-positive/false-negative

rates compromise effectiveness in practice. Zhou et al. [10] provide a more detailed and systematic analysis across 13 vulnerability categories and nine types of analysis. Their findings highlighted the lack of standardized vulnerability taxonomies, which complicates reproducibility and cross-tool comparisons. The lack of standardized definitions hinders instrument development and empirical evaluation, thus limiting the effectiveness of static analysis.

Machine Learning-Based Detection To overcome the rigidity of rule-based analyzers, researchers have increasingly turned to ML. Kumar et al. [11] evaluated machine learning models for detecting DDoS attacks in IoT smart contract systems. Xu et al. [12] studied the detection of eight smart contract vulnerabilities using features derived from abstract syntax trees (ASTs). Both studies reported excellent detection performance with over 90% accuracy. Compared to static tools such as Oyente and SmartCheck, the ML models were more accurate and faster but still had issues finding specific vulnerable lines of code, generalizing outside Ethereum's Solidity language, and extending coverage to new blockchain systems.

Deep Learning-Based Approaches DL has been explored to capture complex semantic patterns in smart contract bytecode. Sendner et al. [13] propose ESCORT as a transfer learning system that detects several vulnerabilities concurrently and can easily be adapted to new vulnerability types with minimal retraining. Their results show scalability and generalizability which fixes a major limitation found in previous ML models. Also, Osei et al. [14] propose WIDENNET that encodes the operational codes (OPCODES) from contract bytecode to catch both simple and complex patterns inside it; tested on real datasets achieves an average accuracy of 83.07% thus showing possibility for better performance tuning using deep neural networks (DNNs). In parallel, researchers have also investigated the security risks arising from software reuse. Chen et al. [15] report most smart contract developers copy paste code snippets without caring about security. They developed SOChecker, a vulnerability detection tool combining symbolic execution with a fine-tuned Llama2 model which outperforms large language models such as GPT-3.5 and GPT-4 in detecting threats within incomplete code snippets. This highlights the growing importance to integrate AI-assisted methods into actual developer workflows. Also, Chen et al. [16] conducted an empirical study to evaluate the effectiveness of large language models (GPT-3.5-turbo, GPT-4, and GPT-4o) in detecting vulnerabilities in smart contracts. The authors evaluated the models' capacity to detect nine categories of vulnerabilities based on the DASP10 taxonomy using the Smartbugs-curated reference dataset, which includes 142 Solidity contracts. The study compares the performance of ChatGPT with that of 14 static tools. The main results indicate that, although ChatGPT outperforms traditional tools in terms of speed and detection of complex flaws such as front running or denial of service, it suffers from low precision (around 20–22%) despite a high recall rate (reaching 88.2% for GPT-4). The authors also highlight critical limitations regarding the uncertainty of responses (instability of results on 42% of the contracts tested) and the context length constraint for analyzing large code.

The summary of related work (Table 1) demonstrates important advances, but also significant limitations. Static tools, while widely used, suffer from inconsistent accuracy and lack standardized vulnerability definitions. Machine learning methods have shown promising detection rates but are often limited to Ethereum's Solidity technology and limited datasets. Deep learning frameworks such as ESCORT and WIDENNET improve generalizability and adaptability but remain computationally expensive and have yet to demonstrate consistent superiority on diverse real-world datasets. Furthermore, little attention has been paid to the systematic comparison and binary detection of traditional machine learning models with deep learning approaches in unified experimental contexts.

Table 1. Summary of related work on smart contract vulnerability detection.

Paper	Year	Dataset	Samples	Judgment Tools/Algorithm	Problem Type	Main Result
Durieux et al. [8]	2020	SmartBugs Curated, SmartBugs Wild	47,587	HoneyBadger, Maian, Manticore, Mythril, Osiris, Oyente, Securify, Slither, SmartCheck	Multidetection	Only 42% of vulnerabilities detected.
Ghaleb et al. [9]	2020	Own dataset	50	Oyente, Securify, Mythril, SmartCheck, Manticore, Slither	Multidetection	High rates of false positives and false negatives.
Zhou et al. [10]	2022	–	–	SmartCheck, DefectChecker, ContractWard, NPChecker, MadMax, Osiris, ContractFuzzer, Sereum, sFuzz	Multidetection	Detection limited to predefined vulnerability patterns.
Kumar et al. [11]	2021	BoT-IoT	2,934,817	Random Forest, XGBoost	Multidetection	Random Forest outperforms XGBoost.
Xu et al. [12]	2021	SmartBugs, SolidiFi, SmartBugsWild	408	Oyente, SmartCheck; KNN; SGD	Multidetection	KNN achieves superior performance.
Sendner et al. [13]	2023	Own dataset	3,640,153	ESCORT (Multilayer Perceptron)	Multidetection	Transfer learning enhances detection of new vulnerabilities.
Osei et al. [14]	2024	–	–	WIDENNET (MLP-based)	Binary classification	WIDENNET reaches 83.07% accuracy.
Chen et al. [15]	2024	Own dataset	897	SOChecker (fine-tuned Llama-2)	Multidetection	SOChecker Outperforms GPT-3.5/4 with F1-score of 68.2%.
Chen et al. [16]	2025	SmartBugs Curated	142	ChatGPT (GPT-3.5-turbo, GPT-4, and GPT-4o)	Multidetection	ChatGPT has a high recall rate but low precision depending on the vulnerabilities.

In this context, our work addresses this gap by conducting a comparative study of machine learning and deep learning methods for detecting vulnerabilities in smart contracts. More specifically, we investigated models such as RF, KNN, MLP and CNN-1D, thereby providing a balanced empirical evaluation of classical and modern paradigms. The main stages involved in constructing and evaluating our approach are illustrated in Figure 1.

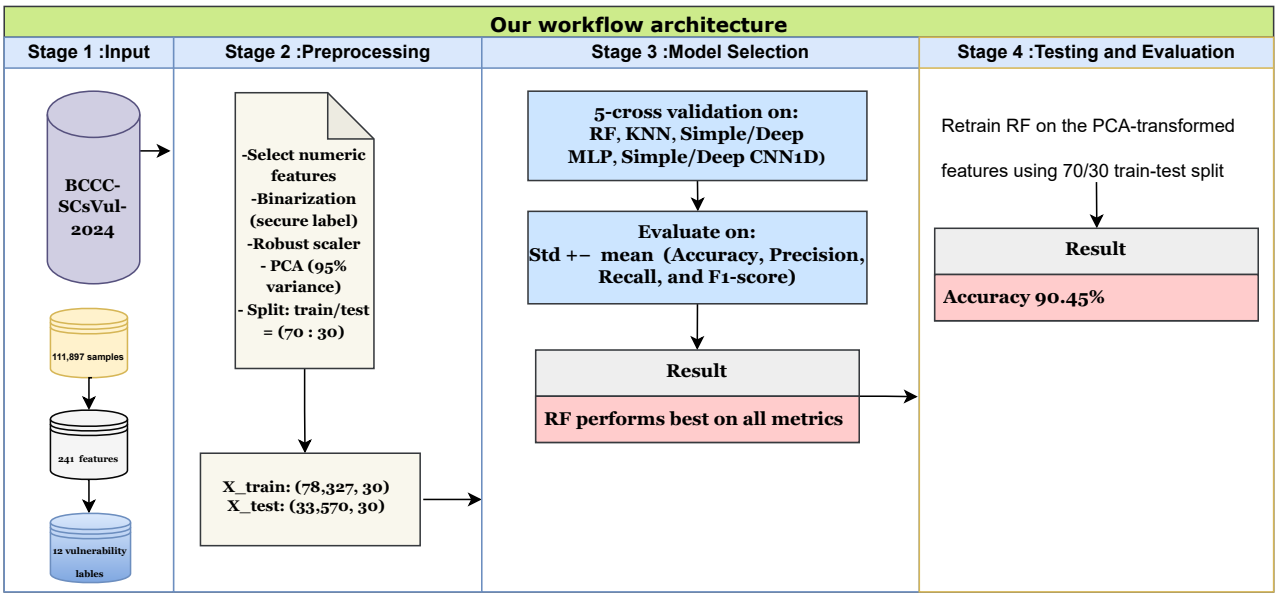

Figure 1. Workflow of our proposed approach.

3. Dataset Description

The BCCC-ScsVuls-2024 dataset [17] was created to compensate for the lag in large, multi-label benchmarks for smart contract detection and profiling. It is derived from Solidity smart contracts gathered from several reputable sources, including SmartBugs, the Ethereum Smart Contract (ESC) collection, Slither-audited contracts, and the SmartScan

dataset. During preprocessing, each contract is transformed into its SHA-256 hash representation. The contracts are then statically analyzed and encoded as numerical feature vectors using the SCsVulLyzer V2.0 framework.

The 2023 version contained 36,670 samples with 70 features and a binary output (0 secure, 1 vulnerable). The 2024 dataset offers both an increase and diversity in features (241 features), the number of classes (12 classes), and the number of samples (111,897). The features are divided into several broad categories. They enable the analysis of smart contracts at several levels of abstraction: (1) There are bytecode-based metrics: statistical measures computed from the compiled code, such as entropy or character-frequency distributions, which can reveal obfuscation or unusually complex logic. (2) Opcode-level statistics record how often key Ethereum Virtual Machine instructions, such as CALL, REVERT, PUSH, JUMP, or SELFDESTRUCT, occur in the bytecode, thereby capturing the behavioral aspects of contract execution. (3) AST features describe the structural composition of the contract by counting node types, exported symbols, and child nodes; these features support higher-level reasoning about program structure. (4) Interface-level (ABI) features measure the lengths and types of input and output definitions, constants, and mutability qualifiers, allowing potential anomalies in function signatures to be detected. (5) Source-code metrics include counts of loops, conditional statements, external calls, and public functions; they also record the distribution of code, blank, and comment lines and track the use of risk-prone Solidity keywords (e.g., delegatecall, send, or create2). (6) Contract metadata such as names, inheritance hierarchies, and the number of functions or events provides context that can be useful for classification and profiling.

The largest vulnerability classes are Re-entrancy (17,698 contracts) and Integer underflow/overflow (16,740 contracts), followed by Denial of Service (12,394), Call to Unknown (11,131), Gas Exception (6879), and Mishandled Exception (5154). Smaller categories include Timestamp (2674), Transaction order dependence (3562), Unused return (3229), Weak access modifier (1918), and External bug (3604). Additionally, the dataset contains 26,914 contracts marked as Secure, indicating that they exhibit none of the listed vulnerabilities.

4. Data Preprocessing and Model Selection

4.1. Data Preprocessing

During the pre-processing phase, an analysis of the features distributions was performed (Figure 2) to assess the presence and extent of outliers in the data set. We selected 238 numerical attributes. Visual inspection revealed a large number of outliers for most attributes such as the count of opcodes and the number of bytecode characters, long-tailed distributions much greater than the interquartile range (IQR). These outliers can significantly disrupt the scaling process when applying traditional normalization techniques.

A comparative analysis was conducted to study the impact of scaling on dimensionality reduction using four preprocessors: No Scaler, StandardScaler, Min-MaxScaler and RobustScaler with PCA. As illustrated in Figure 3, the cumulative explained variance curves demonstrate significant variations in the number of components required to retain 95% of the total variance: a single component with no scaling, 96 components with StandardScaler, 29 components with MinMaxScaler, and 30 components with RobustScaler.

These results indicate that the presence of outliers significantly affects the variance structure of the data, ultimately influencing PCA performance. While StandardScaler normalizes data using the mean and standard deviation, it remains vulnerable to outliers, which can inflate variance and hinder effective dimensionality reduction. On the other hand, RobustScaler uses the median and interquartile range for rescaling, effectively diminishing

the impact of extreme values and yielding a more stable and representative feature space. This approach is robust against outliers [18].

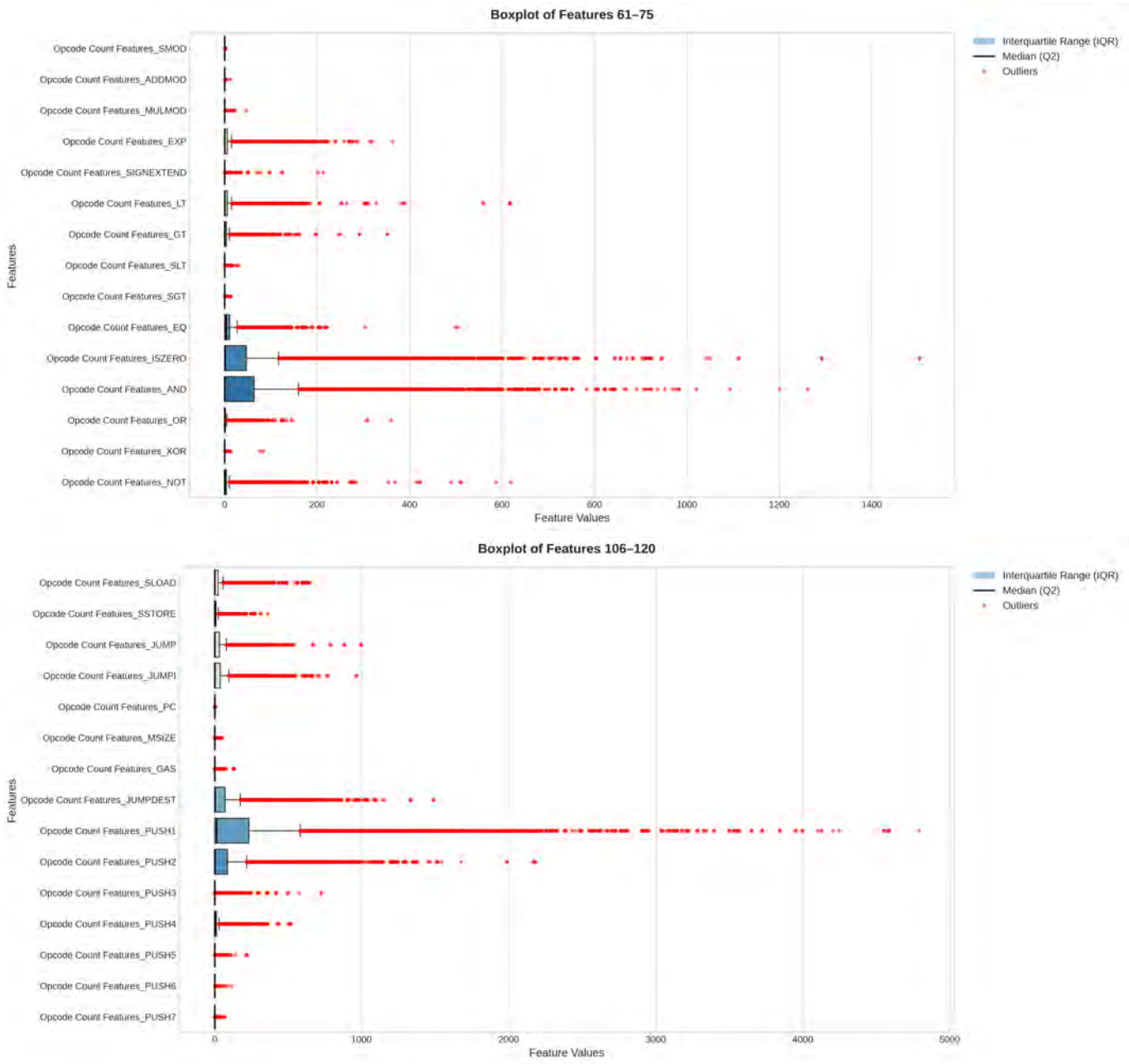

Figure 2. Distribution of some Selected Features 61–75 and 106–120.

Thus, RobustScaler was chosen as the most suitable preprocessing method because it maintains the important features of the data and also reduces the effect of outliers. This preprocessing technique guarantees that subsequent models are trained on unbiased feature distributions, resulting in enhanced reliability and generalizability of learning performance.

For the output class we selected 12th label 'secure' to perform binary vulnerability detection. In the original BCCC-SCsVul-2024 dataset, the authors assigned 1 for secure samples and 0 for vulnerable samples. To synchronize the dataset with our binary classification task, where 1 signifies a vulnerable (positive) sample and 0 denotes a non-vulnerable/secure (negative) sample, we inverted the labels by assigning secure samples a value of 0 and vulnerable samples a value of 1. The converted data was split into training

set (70%) and testing set (30%) using stratified sampling to maintain the class distribution between secure and vulnerable contracts.

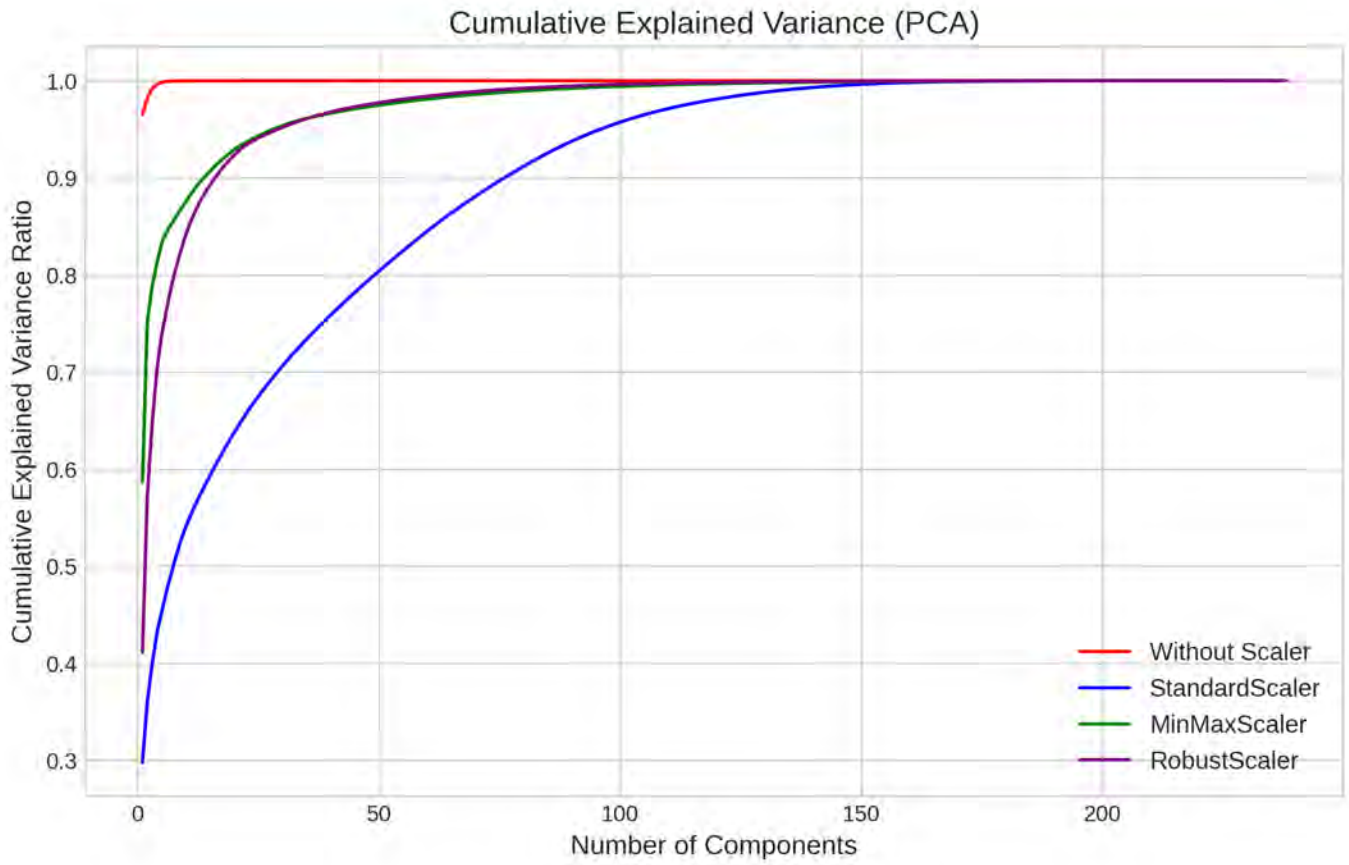

Figure 3. Cumulative explained variance ratio obtained from PCA using different data scaling methods.

4.2. Model Selection

To systematically evaluate the effectiveness of both traditional and deep learning models in detecting smart contract vulnerability, six models were implemented under a unified experimental framework. The model architectures and experimental setup are summarized in Tables 2–4. The selected models include two traditional machine learning algorithms RF and KNN and four deep neural network architectures: SimpleMLP, DeepMLP, SimpleCNN1D, and DeepCNN1D.

Random Forest was chosen due to its robustness against noise and its ability to capture nonlinear interactions through the aggregation of multiple decision trees [19,20]. Its ensemble nature provides inherent regularization, reducing the possibility of overfitting in high-dimensional settings such as smart contract feature representations [21]. We also chose KNN for both its high classification performance and its simplicity among machine learning techniques [22].

In parallel, four deep learning architectures were designed to explore the representation power of neural models on structured feature vectors. The MLP architectures aim to capture complex, nonlinear dependencies among features [23]. The "simple" and "deep"variants differ in their network depth and number of hidden units, enabling analysis of the trade-off between architectural complexity and generalization. The CNN1D were introduced to exploit potential sequential patterns or local correlations within the PCA-transformed feature vectors (adjacent principal components) [24]. The convolutional layers perform feature extraction, while the global max-pooling layer ensures dimensionality reduction and translation invariance.

Table 2. Summary of deep learning model architectures used in the study.

Model	Architecture Description
SimpleMLP	Input layer → Dense(64, ReLU) → Dropout(0.3) → Dense(32, ReLU) → Output layer (Sigmoid).
DeepMLP	Input layer → Dense(128, ReLU) → Dropout(0.3) → Dense(64, ReLU) → Dropout(0.3) → Dense(32, ReLU) → Dense(16, ReLU) → Output layer (Sigmoid).
SimpleCNN1D	Conv1D(32 filters, kernel size 3, ReLU) → GlobalMaxPooling1D → Dense(32, ReLU) → Output layer (Sigmoid).
DeepCNN1D	Conv1D(64 filters, kernel size 3, ReLU) → Conv1D(32 filters, kernel size 3, ReLU) → GlobalMaxPooling1D → Dense(64, ReLU) → Dense(32, ReLU) → Output layer (Sigmoid).

Table 3. Machine learning algorithms and their main configuration parameters.

Model	Main Parameters
RF	Number of estimators: 100; Criterion: Gini; Random state: 42.
KNN	Number of neighbors: 5; Distance metric: Euclidean; Weighting: Uniform.

Table 4. Experimental configuration.

Parameter	Description
Dataset	BCCC-SCsVul-2024 (111,897 samples, 238 features).
Preprocessing	RobustScaler normalization; PCA retaining 95% variance (30 principal components).
Train/Test Split	70% training–30% testing; stratified sampling.
Cross-Validation	5-fold stratified with shuffle.
Optimization Algorithm	Adam optimizer (learning rate = 0.001).
Loss Function	Binary cross-entropy.
Epochs / Batch Size	15 epochs; batch size = 128.
Evaluation Metrics	Accuracy, Precision, Recall, F1-score (mean ± std).
Hardware Environment	Executed in free Google Colab (NVIDIA Tesla T4 GPU (15 GB VRAM), 12.7 GB of system RAM Intel Xeon @ 2.00 GHz, with 2 cores and 2 threads).
Library versions	TensorFlow: 2.19.0/Scikit-learn: 1.6.1.

All models have been evaluated using a five-fold stratified cross-validation procedure to maintain class balance among folds and to ensure statistical rigor. For each fold, the following metrics were calculated: accuracy, precision, recall, and F1-score. Results are reported as mean ± standard deviation, expressing both main tendency and dispersion (consistency) across validation splits. Following cross-validation, the best model was retrained on the training set of principal components and finally tested on the test set. A confusion matrix of robust models was then constructed to provide class-specific performance and identify potential sources of misclassifications, such as false positives or missed weaknesses.

5. Results

5.1. Overall Models

The cross-validation results summarized in Table 5 and Figure 4 highlight clear yet consistent performance differences among the six evaluated models.

Table 5. Cross-validation performance (mean ± std) of selected models.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
RF	**89.44 ± 0.10**	**91.29 ± 0.11**	**95.18 ± 0.22**	**93.20 ± 0.07**
KNN	85.13 ± 0.28	88.65 ± 0.19	92.22 ± 0.29	90.40 ± 0.18
DeepMLP	82.83 ± 0.05	86.98 ± 0.33	91.03 ± 0.45	88.96 ± 0.05
DeepCNN1D	81.69 ± 0.39	84.82 ± 1.47	92.52 ± 2.11	88.47 ± 0.31
SimpleMLP	81.95 ± 0.32	85.27 ± 1.14	92.22 ± 1.61	88.58 ± 0.24
SimpleCNN1D	79.13 ± 0.35	81.60 ± 1.30	93.74 ± 2.70	87.21 ± 0.48

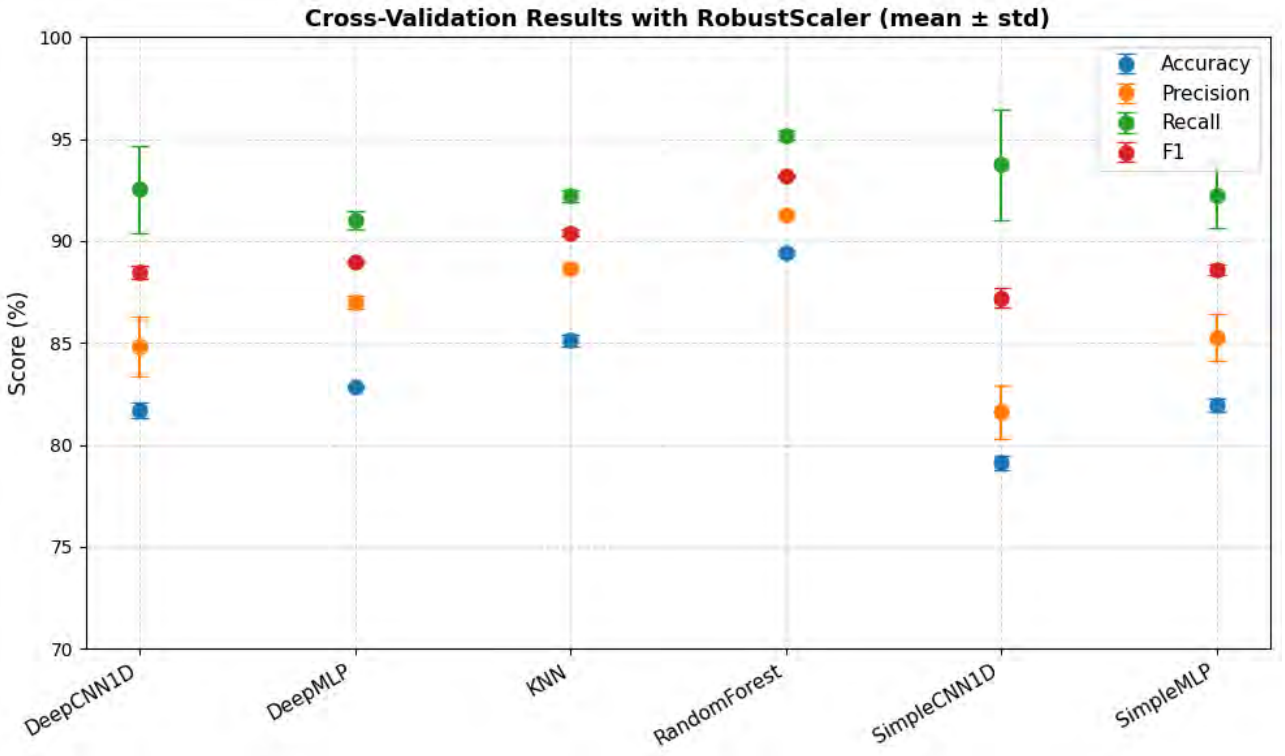

Figure 4. Cross validation performance of selected models.

Across all models, standard deviations remain relatively small (below 2.7% for every metric), indicating stable cross-fold performance and confirming the reliability of the validation procedure. The smallest standard deviations, confirming strong consistency across cross-validation folds. While larger variability indicates greater sensitivity to data partitioning and stochastic training effects. Despite these shared stability characteristics, the performance metrics vary considerably depending on the complexity of the model.

Random Forest performs best on all metrics, with accuracy, precision, recall, and F1 score above 89%. It benefits from ensemble averaging, which reduces variance and captures complex feature interactions. KNN is the second-best model; its instance-based approach yields competitive recall (92.22 ± 0.29) and F1 scores (90.40 ± 0.18) but with lower precision (88.65 ± 0.19) and accuracy (85.13 ± 0.28) than RF. Both MLPs and CNNs show moderate performance, indicating that the available training samples and the reduced features obtained through PCA may not fully exploit the capabilities of these deep architectures.

For all models, recall is higher than precision. This indicates that classifiers are more sensitive to the vulnerable class and tend to flag vulnerable contracts rather than miss them. However, models with lower precision (especially Simple CNN1D and Deep CNN1D) produce more false positives, which could lead to unnecessary code audits in practice.

The deeper versions of the neural networks (Deep MLP and Deep CNN1D) outperform their simpler peers, showing that additional layers help capture more complex patterns in the data. Nevertheless, even the deep architecture does not surpass traditional models. This suggests that the combination of RobustScaler + PCA may reduce feature complexity to a level where simpler algorithms like RF and KNN are more effective, while deeper networks risk overfitting or under-utilizing their capacity.

Using the Kruskal–Wallis test [25], we compared the performance of the different models included in our study to determine whether statistically significant differences existed among them. All obtained p-values were below the 0.05 significance threshold (Accuracy: 0.000045, F1-score: 0.000056, Precision: 0.000095, Recall: 0.010847). Therefore, we reject the null hypothesis that all models perform equally. These results confirm that the performance differences observed between models are statistically significant.

The results in the Table 6 show marked differences in computational cost between the evaluated models. Classical algorithms are the least resource-intensive: KNN has negligible training time, while random forest requires longer execution time due to the construction of multiple decision trees. Deep learning models are naturally more expensive, with training times ranging from approximately 32 to 45 s depending on the network depth, reflecting the iterative nature of gradient descent optimization. Memory usage remains modest for all models: RAM consumption remains below 3 GB, and only neural networks utilize the GPU. Among these, MLPs allocate approximately 124 to 127 MB of GPU memory, while CNN-based architectures use more (up to 326 MB) due to convolutional operations. Overall, the measurements confirm that, although deep neural models are more computationally intensive than classical methods, all approaches remain lightweight enough to be trained comfortably within the free limits of Google Colab's GPU. Table 7 summarizes the evaluated models in terms of strengths, weaknesses, complexity (training and prediction), and stability under PCA.

Table 6. Average training time and memory usage for each model.

Model	Avg. Training Time (s)	Avg. RAM Usage (MB)	Avg. GPU Usage (MB)
RF	51.60	1921.97	0
KNN	0.003	1924.57	0
DeepMLP	41.80	2425.36	127.20
DeepCNN1D	44.99	2715.31	326.00
SimpleMLP	32.41	2370.62	124.00
SimpleCNN1D	34.78	2560.03	198.00

All evaluated models demonstrate satisfactory stability and generalization, but RF consistently provides the highest accuracy, recall, and F1-score, making it the most reliable and well-balanced model for binary vulnerability detection. The low std model shows strong stability by exhibiting low dispersion in measurements, making the RF robust.

5.2. PCA-Based Feature Contribution Analysis

Our choice was motivated by its superior performance and stability across all metrics. For that, the RF was retrained on the PCA-transformed features using a 70/30 train-test split. The classifier achieved a test accuracy of 90.45%, confirming the consistency observed during cross-validation. In our context, where 1 denotes a vulnerable contract and 0 a

secure one, the confusion matrix (Table 8) provides an explicit interpretation of the model's decisions. A true positive (TP) (24,411 cases) corresponds to a vulnerability correctly detected, while a true negative (TN) (5952 cases) indicates that a safe contract was accurately identified. A false positive (FP) (2122 cases) occurs when a non-vulnerable contract is incorrectly flagged as vulnerable, which increases unnecessary manual inspections but does not pose a direct security threat. The most critical outcome is the false negative (FN) (1058 cases), where an actual vulnerability goes undetected, potentially exposing the system to attacks. Reducing false negatives is therefore essential for security, whereas minimizing false positives improves operational efficiency.

Table 7. Comparison of machine learning models in terms of strengths, weaknesses, complexity, and stability under PCA.

Model	Strengths	Weaknesses	Complexity (Training/Prediction)	Stability Under PCA
RF	High accuracy and F1; ensemble averaging captures complex feature interactions; robust to noise	Requires longer training time (≈51 s); relies on multiple decision trees	Moderate training cost; fast prediction	High—best performance after PCA with low variance across folds
KNN	Simple training; non-parametric model	Requires storing the entire dataset; sensitive to high dimensionality	Low training cost; high prediction cost	Moderate—PCA helps remove noisy dimensions, but stability remains lower than RF
SimpleMLP	Captures non-linear patterns; moderate resource usage (≈32 s, ≈124 MB GPU)	Lower accuracy (≈82%); reduced feature space may under-utilise model capacity	Moderate	Moderate—stable performance but no clear improvement with PCA
DeepMLP	Deeper architecture captures more complex relationships; slightly improved metrics over SimpleMLP	Higher risk of overfitting and increased computational cost; still inferior to RF	Higher training time (≈41 s) and GPU usage (≈127 MB)	Moderate—deeper models are more stable than shallow ones, but PCA may remove useful feature structure
SimpleCNN 1D	Exploits complex local patterns; modest training time (≈34 s)	Lowest accuracy (79%) and precision; high GPU usage (≈198 MB); sensitive to noisy features	High—convolution operations increase computational cost	Low—performance degrades after PCA with higher variance across folds
DeepCNN 1D	Multiple convolutional layers capture complex sequential patterns	Low precision (84.82%) and accuracy (81.69%); highest GPU usage (≈326 MB)	Highest complexity among tested models; heavy convolution operations	Low—PCA removes local structure, leading to larger performance variance

Table 8. Confusion Matrix for the Random Forest Classifier.

	Predicted Non-Vulnerable	Predicted Vulnerable
True Non-vulnerable	5952	2122
True Vulnerable	1085	24,411

All evaluated models exhibit higher recall than precision (Table 5), reflecting a deliberate design choice favoring vulnerability sensitivity over false alarm reduction. This behavior is desirable in security-critical contexts, where missing a vulnerability may lead to severe financial or operational consequences. Remaining false negatives are likely associated with rare, low-frequency, or structurally subtle vulnerability patterns, which are more difficult to capture in a binary classification setting.

We then examined the contribution of the original features to the first ten principal components to identify the primary sources of variance (Figure 5). The first 4 components showed strong contributions from opcode features such as PUSH, DUP, SWAP, and RETURNDATASIZE, which correspond to stack and memory manipulation mechanisms. These low-level execution instructions account for the primary behavioral differences between vulnerable and non-vulnerable contracts. The subsequent components reflected more structural characteristics, including code length, duplicate line count, and cryptographic operations (SHA3, ADDRESS, and INVALID), which capture aspects of software complexity and error handling. This hierarchy of contributions suggests that behavioral variability is mainly driven by execution semantics rather than syntactic or structural code features. The opcode-related features dominate the first components, while code complexity and entropy metrics appear mainly in the later ones. Solidity-specific and ABI-related variables showed minimal influence, suggesting that vulnerability detection is primarily governed by execution-level instructions rather than high-level syntactic descriptors.

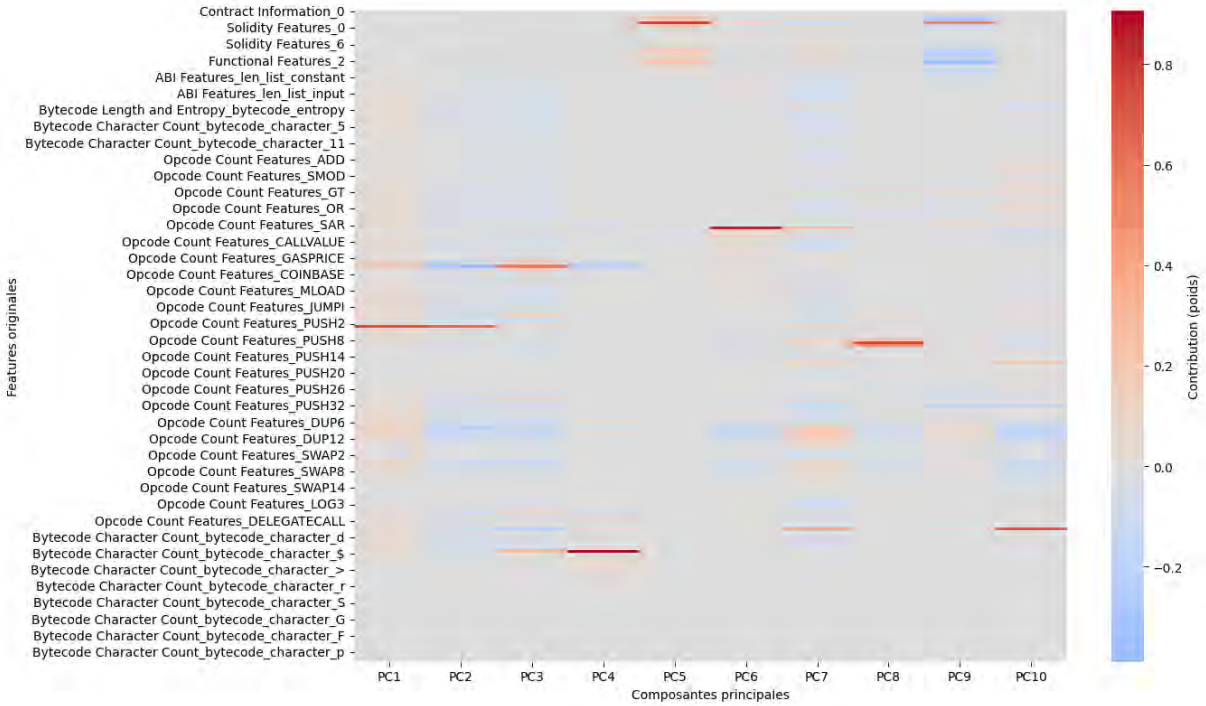

Figure 5. Feature contribution heatmap for the first ten PCA components.

6. Conclusions

The current study provided a systematic comparison between the traditional machine learning models (RF and KNN) and deep learning models (SimpleMLP, DeepMLP, SimpleCNN1D, and DeepCNN1D) on vulnerability detection of smart contracts from the BCCC-SCsVuls-2024 dataset. After rigorous preprocessing with RobustScaler and PCA, six models were trained and validated under identical conditions. Our comparative evaluation demonstrated that Random Forest outperformed all other models in terms of accuracy, precision, recall, and F1-score, confirming its robustness and stability across validation folds. Deep architectures such as MLP and CNN-1D achieved competitive recall but showed higher variability, indicating sensitivity to the dimensionality reduction process and training data partitioning.

A PCA-based analysis further revealed that vulnerability binary detection is largely governed by low-level opcode features related to stack and memory management, whereas higher-level solidity and structural metrics had limited influence. This highlights the

importance of execution semantics in driving vulnerability behavior and validates the relevance of feature-level interpretability in vulnerability detection frameworks.

In future research, we plan to expand this study in two directions. First, multi-label classification will be introduced to simultaneously detect multiple vulnerability types within the same contract. Second, we aim to build an automated, real-time detection pipeline deployable in development environments, enhancing proactive security auditing for blockchain developers.

Author Contributions: Conceptualization, M.Y.A., W.H.G. and S.W.K.; methodology, M.Y.A., W.H.G. and S.W.K.; software, M.Y.A., W.H.G. and S.W.K.; validation, M.F.; writing—original draft preparation, M.Y.A., W.H.G. and S.W.K.; writing—review and editing, M.F.; supervision, M.F.; project administration, M.F. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by The Junior Professor Chair (CPJ) of Franche Comte University, Galaxie number 4718 ANR-23-CPJ1-0010-01.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data we used is public. The dataset can be request via: https://www.yorku.ca/research/bccc/ucs-technical/cybersecurity-datasets-cds/smart-contracts-vulnerabilities-bccc-scsvuls-2024/ (accessed on 14 December 2025).

Acknowledgments: Authors are grateful to the CPJ project (ANR-23-CPJ1-0010-01). Authors are grateful to the Researchers Supporting Project (ANUI/2025/ENG28), Alnoor University, Mosul, Iraq.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Tan, T.M.; Saraniemi, S. Trust in blockchain-enabled exchanges: Future directions in blockchain marketing. *J. Acad. Mark. Sci.* **2023**, *51*, 914–939. [CrossRef]
2. Abdelhamid, M.; Hassan, G. Blockchain and smart contracts. In Proceedings of the 8th International Conference on Software and Information Engineering, Cairo, Egypt, 9–12 April 2019; pp. 91–95.
3. Szabo, N. Formalizing and securing relationships on public networks. *First Monday* **1997**, *2*. [CrossRef]
4. Clack, C.D.; Bakshi, V.A.; Braine, L. Smart contract templates: Foundations, design landscape and research directions. *arXiv* **2016**, arXiv:1608.00771.
5. Tang, X.; Du, Y.; Lai, A.; Zhang, Z.; Shi, L. Deep learning-based solution for smart contract vulnerabilities detection. *Sci. Rep.* **2023**, *13*, 20106. [CrossRef] [PubMed]
6. De Baets, C.; Suleiman, B.; Chitizadeh, A.; Razzak, I. Vulnerability detection in smart contracts: A comprehensive survey. *arXiv* **2024**, arXiv:2407.07922. [CrossRef]
7. Jiang, F.; Chao, K.; Xiao, J.; Liu, Q.; Gu, K.; Wu, J.; Cao, Y. Enhancing smart-contract security through machine learning: A survey of approaches and techniques. *Electronics* **2023**, *12*, 2046. [CrossRef]
8. Durieux, T.; Ferreira, J.F.; Abreu, R.; Cruz, P. Empirical review of automated analysis tools on 47,587 ethereum smart contracts. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Seoul, Republic of Korea, 27 June 2020; pp. 530–541.
9. Ghaleb, A.; Pattabiraman, K. How effective are smart contract analysis tools? evaluating smart contract static analysis tools using bug injection. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual, 18–22 July 2020; pp. 415–427.
10. Zhou, H.; Milani Fard, A.; Makanju, A. The state of ethereum smart contracts security: Vulnerabilities, countermeasures, and tool support. *J. Cybersecur. Priv.* **2022**, *2*, 358–378. [CrossRef]
11. Kumar, P.; Kumar, R.; Gupta, G.P.; Tripathi, R. A Distributed framework for detecting DDoS attacks in smart contract-based Blockchain-IoT Systems by leveraging Fog computing. *Trans. Emerg. Telecommun. Technol.* **2021**, *32*, e4112. [CrossRef]
12. Xu, Y.; Hu, G.; You, L.; Cao, C. A novel machine learning-based analysis model for smart contract vulnerability. *Secur. Commun. Netw.* **2021**, *2021*, 5798033. [CrossRef]

13. Sendner, C.; Chen, H.; Fereidooni, H.; Petzi, L.; König, J.; Stang, J.; Dmitrienko, A.; Sadeghi, A.R.; Koushanfar, F. Smarter Contracts: Detecting Vulnerabilities in Smart Contracts with Deep Transfer Learning. In Proceedings of the NDSS, San Diego, CA, USA, 27 February–3 March 2023.
14. Osei, S.B.; Ma, Z.; Huang, R. Smart contract vulnerability detection using wide and deep neural network. *Sci. Comput. Program.* **2024**, *238*, 103172. [CrossRef]
15. Chen, J.; Chen, C.; Hu, J.; Grundy, J.; Wang, Y.; Chen, T.; Zheng, Z. Identifying smart contract security issues in code snippets from stack overflow. In Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, Vienna, Austria, 16–20 September 2024; pp. 1198–1210.
16. Chen, C.; Su, J.; Chen, J.; Wang, Y.; Bi, T.; Yu, J.; Wang, Y.; Lin, X.; Chen, T.; Zheng, Z. When chatgpt meets smart contract vulnerability detection: How far are we? *ACM Trans. Softw. Eng. Methodol.* **2025**, *34*, 1–30. [CrossRef]
17. HajiHosseinKhani, S.; Lashkari, A.H.; Oskui, A.M. Unveiling smart contract vulnerabilities: Toward profiling smart contract vulnerabilities using enhanced genetic algorithm and generating benchmark dataset. *Blockchain Res. Appl.* **2025**, *6*, 100253. [CrossRef]
18. Ramsauer, A.; Baumann, P.M.; Lex, C. The Influence of Data Preparation on Outlier Detection in Driveability Data. *SN Comput. Sci.* **2021**, *2*, 222. [CrossRef]
19. Salman, H.A.; Kalakech, A.; Steiti, A. Random forest algorithm overview. *Babylon. J. Mach. Learn.* **2024**, *2024*, 69–79. [CrossRef] [PubMed]
20. Buskirk, T.D. Surveying the forests and sampling the trees: An overview of classification and regression trees and random forests with applications in survey research. *Surv. Pract.* **2018**, *11*. [CrossRef]
21. Halabaku, E.; Bytyçi, E. Overfitting in Machine Learning: A Comparative Analysis of Decision Trees and Random Forests. *Intell. Autom. Soft Comput.* **2024**, *39*, 987. [CrossRef]
22. Cunningham, P.; Delany, S.J. K-nearest neighbour classifiers-a tutorial. *ACM Comput. Surv. (CSUR)* **2021**, *54*, 1–25. [CrossRef]
23. Grosse, R. Lecture 5: Multilayer Perceptrons. *Inf. Tec.* **2019**. Available online: https://www.cs.toronto.edu/~bonner/courses/2020f/csc311/lectures/nn_notes.pdf (accessed on 14 December 2025).
24. Ige, A.O.; Sibiya, M. State-of-the-art in 1d convolutional neural networks: A survey. *IEEE Access* **2024**, *12*, 144082–144105. [CrossRef]
25. Kruskal, W.H.; Wallis, W.A. Use of ranks in one-criterion variance analysis. *J. Am. Stat. Assoc.* **1952**, *47*, 583–621. [CrossRef]